Database e linguaggio SQL

Indice

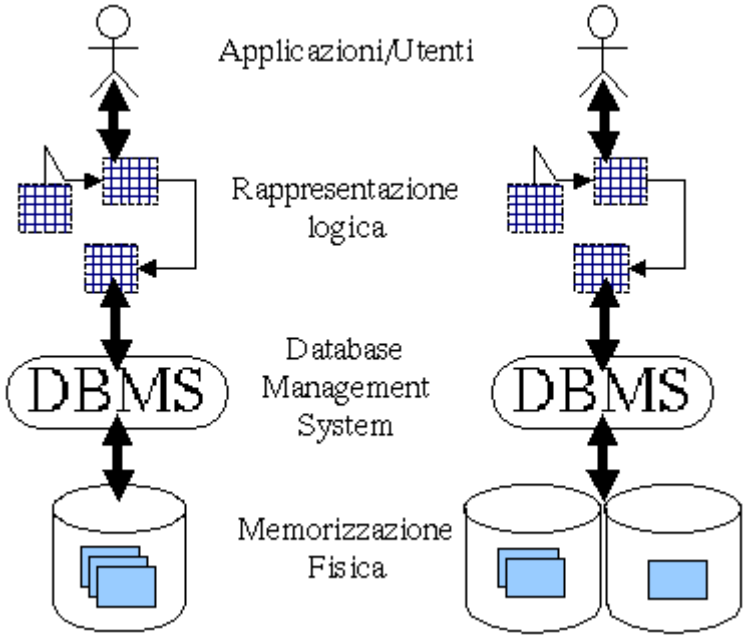
1. Web, Database e DBMS

Il World Wide Web e' forse una delle maggiori fonti di informazione a cui oggi possiamo attingere: avendo a disposizione un collegamento a Internet ed un browser Web, un software ormai comune su qualsiasi computer, abbiamo la possibilita' di consultare un patrimonio di centinaia di milioni di pagine riguardanti praticamente ogni argomento di interesse.

Spesso queste pagine non sono documenti statici, ma vengono creati dinamicamente quando noi li richiediamo e le informazioni che contengono vengono estratte da un database. Se il database e' un database **relazionale** (vedremo in seguito cosa questo significa), probabilmente il linguaggio utilizzato per recuperare le informazioni che ci vengono mostrate e' SQL (Structured Query Language).

Prima di occuparci di cos'e' e come si usa SQL cerchiamo di capire cosa si intende con la parola database, che spesso in italiano viene tradotta come "base di dati".
Un database e' una collezione di dati che viene gestita e organizzata da un software specifico, il DBMS (DataBase Management System, Sistema di Gestione di DataBase). Un DBMS e' sostanzialmente uno strato software che si frappone fra l'utente ed i dati veri e propri. Grazie a questo strato intermedio l'utente e le applicazioni non accedono ai dati cosi' come sono memorizzati effettivamente, cioe' alla loro rappresentazione fisica, ma ne vedono solamente una rappresentazione logica. Cio' permette un elevato grado di indipendenza fra le applicazioni e la memorizzazione fisica dei dati. L'amministratore del database, se ne sente la necessita', puo' decidere di memorizzare i dati in maniera differente o anche di cambiare il DBMS senza che le applicazioni, e quindi gli utenti, ne risentano. La cosa importante e' che non venga cambiata la rappresentazione logica di quei dati, che e' la sola cosa che i loro utilizzatori conoscono. Questa rappresentazione logica viene chiamata 'Schema del database' ed e' la forma di rappresentazione dei dati piu' a basso livello a cui un utente del database puo' accedere. Ad esempio, in Figura 1 e' rappresentata una situazione in cui l'amministratore del database ha deciso che per motivi di efficienza era necessario cambiare il disco su cui erano memorizzati alcuni dati, partizionandoli

inoltre su piu' dischi per permettere accessi paralleli a sottoinsiemi di dati indipendenti. Dal punto di vista dell'utente non e' cambiato assolutamente nulla e probabilmente egli non e' nemmeno a conoscenza dell'avvenuto cambiamento.



La caratteristica principale secondo cui i DBMS vengono normalmente classificati e' appunto la rappresentazione logica dei dati che essi mostrano ai loro utilizzatori. Nel corso degli anni sono stati adottati numerosi modelli per i dati, a fronte dei quali esistono quindi vari tipi di database. I tipi piu' comuni sono:

Database gerarchici: i dati vengono organizzati in insiemi legati fra loro da relazioni di "possesso", in cui un insieme di dati puo' possedere altri insiemi di dati, ma un insieme puo' appartenere solo ad un altro insieme. La struttura risultante e' un albero di insiemi di dati.

Database reticolari: il modello reticolare e' molto simile a quello gerarchico, ed infatti nasce come estensione di quest'ultimo. Anche in questo modello insiemi di dati sono legati da relazioni di possesso, ma ogni insieme di dati puo' appartenere a uno o piu' insiemi. La struttura risultante e' una rete di insiemi di dati.

Database relazionali: i database appartenenti a questa categoria si basano sul modello relazionale la cui struttura principale e' la relazione, cioe' una tabella bidimensionale composta da righe e colonne. Ciascuna riga, che in terminologia relazionale viene chiamata tupla, rappresenta un'entita' che noi vogliamo memorizzare nel database. Le caratteristiche di ciascuna entita' sono definite invece dalle colonne delle relazioni, che vengono chiamate attributi. Entita' con caratteristiche comuni, cioe' descritti dallo stesso insieme di attributi, faranno parte della stessa relazione.

Database ad oggetti (object-oriented): lo schema di un database ad oggetti e' rappresentato da un insieme di classi, che definiscono le caratteristiche ed il comportamento degli oggetti che popoleranno il database. La principale differenza con i modelli esaminati finora e' la non passivita' dei dati. Infatti con un database tradizionale (intendendo con questo termine qualunque database non ad oggetti) le operazioni che devono essere effettuate sui dati vengono demandate alle applicazioni che li utilizzano. Con un database object-oriented, al contrario, gli oggetti memorizzati nel database contengono sia i dati che le operazioni possibili su tali dati. In un certo senso potremmo pensare agli oggetti come a dati a cui e' stata fatta una iniezione di intelligenza, che gli permette di sapere come comportarsi, senza doversi appoggiare ad applicazioni esterne.

I primi due tipi di database, quelli gerarchici e reticolari, quasi appartengono ormai alla storia dell'informatica.
La maggior parte dei database attualmente utilizzati appartiene alla categoria dei database relazionali. I motivi di questo successo (anche commerciale) vanno ricercati nella rigorosita' matematica e nella potenzialita' espressiva del modello relazionale su cui si basano, nella sua semplicita' di utilizzo e, ultima ma non meno importante, nella disponibilita' di un linguaggio di interrogazione standard, l'SQL, che, almeno potenzialmente, permette di sviluppare applicazioni independenti dal particolare DBMS relazionale utilizzato.
I database ad oggetti sono la nuova frontiera nella ricerca sui database, infatti le loro caratteristiche di estendibilita', derivanti dalla possibilita' di definire nuovi tipi di dati e comportamenti, li rendono particolarmente appetibili per tutte quelle applicazioni che richiedono dati complessi, come ad esempio immagini, suoni o coordinate. Purtroppo la mancanza di un modello per gli oggetti universalmente accettato e la non disponibilita' di un linguaggio di interrogazione standard fanno si' che ogni produttore implementi la propria visione specifica, di solito assolutamente incompatibile con tutte le altre. Di recente sono apparsi sul mercato alcuni database, definiti object-relational, che cercano di introdurre nel modello relazionale le caratteristiche di estendibilita' proprie dei database object-oriented.

Indipendentemente dal tipo di database, le funzionalita' principali che ci si deve aspettare da un DBMS sono quelle di:

- consentire l'accesso ai dati attraverso uno schema concettuale, invece che attraverso uno schema fisico;
- permettere la condivisione e l'integrazione dei dati fra applicazioni differenti;
- controllare l'accesso concorrente ai dati;
- assicurare la sicurezza e l'integrita' dei dati.

Grazie a queste caratteristiche le applicazioni che vengono sviluppate possono contare su una sorgente dati sicura, affidabile e generalmente scalabile. Tali proprieta' sono auspicabili per applicazioni che usano la rete Internet come infrastruttura e che hanno quindi evidenti problemi di sicurezza e scalabilita'.

2. Il modello relazionale

I database relazionali sono il tipo di database attualmente piu' diffuso. I motivi di questo successo sono fondamentalmente due:

1. forniscono sistemi semplici ed efficienti per rappresentare e manipolare i dati
2. si basano su un modello, quello relazionale, con solide basi teoriche

Il modello relazionale e' stato proposto originariamente da E.F. Codd in un ormai famoso articolo del 1970. Grazie alla sua coerenza ed usabilita', **il modello e' diventato negli anni '80 quello piu' utilizzato per la produzione di DBMS**.
La struttura fondamentale del modello relazionale e' appunto la "relazione", cioe' una tabella bidimensionale costituita da righe (tuple) e colonne (attributi). Le relazioni rappresentano le entita' che si ritiene essere interessanti nel database. Ogni istanza dell'entita' trovera' posto in una tupla della relazione, mentre gli attributi della relazione rappresenteranno le proprieta' dell'entita'. Ad esempio, se nel database si dovranno rappresentare delle persone, si potra' definire una relazione chiamata "Persone", i cui attributi descrivono le caratteristiche delle persone (Figura 2). Ciascuna tupla della relazione "Persone" rappresentera' una particolare persona.

Persone				
nome	**cognome**	**data_nascita**	**sesso**	**stato_civile**
Mario	Rossi	29/03/1965	M	Coniugato
Giuseppe	Russo	15/11/1974	M	Celibe
Alessandra	Mondella	13/06/1970	F	Nubile

In realta', volendo essere rigorosi, una relazione e' solo la definizione della struttura della tabella, cioe' il suo nome e l'elenco degli attributi che la compongono. Quando essa viene popolata con delle tuple, si parla di "istanza di relazione". Percio' la precedente Figura 2 rappresenta un'istanza della relazione persona. Una rappresentazione della definizione di tale relazione potrebbe essere la seguente:

Persone (nome, cognome, data_nascita, sesso, stato_civile)

Nel seguito si indicheranno entrambe (relazione ed istanza di relazione) con il termine "relazione", a meno che non sia chiaro dal contesto a quale accezione ci si riferisce.
Le tuple in una relazione sono un insieme nel senso matematico del termine, cioe' una collezione non ordinata di elementi differenti. Per distinguere una tupla da un'altra si ricorre al concetto di "chiave primaria", cioe' ad un insieme di attributi che permettono di identificare univocamente una tupla in una relazione. Naturalmente in una relazione possono esserci piu' combinazioni di attributi che permettono di identificare univocamente una tupla ("chiavi candidate"), ma fra queste ne verra' scelta una sola da utilizzare come chiave primaria. Gli attributi della chiave primaria non possono assumere il valore null (che significa un valore non determinato), in quanto non permetterebbero piu' di identificare una particolare tupla in una relazione. Questa proprieta' delle relazioni e delle loro chiavi primarie va sotto il nome di integrita' delle entita' (entity integrity).

Spesso per ottenere una chiave primaria "economica", cioe' composta da pochi attributi facilmente manipolabili, si introducono uno o piu' attributi fittizi, che conterranno dei codici identificativi univoci per ogni tupla della relazione.
Ogni attributo di una relazione e' caratterizzato da un nome e da un dominio. Il dominio indica quali valori possono essere assunti da una colonna della relazione. Spesso un dominio viene definito attraverso la dichiarazione di un tipo per l'attributo (ad esempio dicendo che e' una stringa di dieci caratteri), ma e' anche possibile definire domini piu' complessi e precisi. Ad esempio per l'attributo "sesso" della nostra relazione "Persone" possiamo definire un dominio per cui gli unici valori validi sono 'M' e 'F'; oppure per l'attributo "data_nascita" potremmo definire un dominio per cui vengono considerate valide solo le date di nascita dopo il primo gennaio del 1960, se nel nostro database non e' previsto che ci siano persone con data di nascita antecedente a quella. Il DBMS si occupera' di controllare che negli attributi delle relazioni vengano inseriti solo i valori permessi dai loro domini.
Caratteristica fondamentale dei domini di un database relazionale e' che siano "atomici", cioe' che i valori contenuti nelle colonne non possano essere separati in valori di domini piu' semplici. Piu' formalmente si dice che non e' possibile avere attributi multivalore (multivalued). Ad esempio, se una caratteristica delle persone nel nostro database fosse anche quella di avere uno o piu' figli, non sarebbe possibile scrivere la relazione Persone nel seguente modo:

Persone (nome, cognome, data_nascita, sesso, stato_civile, figli)

Infatti l'attributo figli e' un attributo non-atomico, sia perche' una persona puo' avere piu' di un figlio, sia perche' ogni figlio avra' varie caratteristiche che lo descrivono. Per rappresentare queste entita' in un database relazionale bisogna definire due relazioni:

Persone(*numero_persona, nome, cognome, data_nascita, sesso, stato_civile)
Figli(*numero_persona, *nome_cognome, eta, sesso)

Nelle precedenti relazioni gli asterischi (*) indicano gli attributi che compongono le loro chiavi primarie. Si noti l'introduzione nella relazione Persone dell'attributo numero_persona, attraverso il

quale si assegna a ciascuna persona un identificativo numerico univoco che viene utilizzato come chiave primaria. Queste relazioni contengono solo attributi atomici. Se una persona ha piu' di un figlio, essi saranno rappresentati in tuple differenti della relazione Figli. Le varie caratteristiche dei figli sono rappresentate dagli attributi della relazione Figli. Il legame fra le due relazioni e' costituito dagli attributi numero_persona che compaiono in entrambe le relazioni e che permettono di assegnare ciascuna tupla della relazione figli ad una particolare tupla della relazione Persone. Piu' formalmente si dice che l'attributo numero_persona della relazione Figli e' una chiave esterna (foreign key) verso la relazione Persone. Una chiave esterna e' una combinazione di attributi di una relazione che sono chiave primaria per un'altra relazione. Una caratteristica fondamentale dei valori presenti in una chiave esterna e' che, a meno che non siano null, devono corrispondere a valori esistenti nella chiave primaria della relazione a cui si riferiscono. Nel nostro esempio cio' significa che non puo' esistere nella relazione Figli una tupla con un valore dell'attributo numero_persona, senza che anche nella relazione Persone esista una tupla con lo stesso valore per la sua chiave primaria. Questa proprieta' va sotto il nome di integrita' referenziale (referential integrity)

Uno dei grandi vantaggi del modello relazionale e' che esso definisce anche un algebra, chiamata appunto "algebra relazionale". Tutte le manipolazioni possibili sulle relazioni sono ottenibili grazie ala combinazione di cinque soli operatori: REStrICT, PROJECT, TIMES, UNION e MINUS. Per comodita' sono stati anche definiti tre operatori addizionali che comunque possono essere ottenuti applicando i soli cinque operatori fondamentali: JOIN, INTERSECT e DIVIDE. Gli operatori relazionali ricevono come argomento una relazione o un insieme di relazioni e restituiscono una singola relazione come risultato.
Vediamo brevemente questi otto operatori:

REStrICT: restituisce una relazione contenente un sottoinsieme delle tuple della relazione a cui viene applicato. Gli attributi rimangono gli stessi.

PROJECT: restituisce una relazione con un sottoinsieme degli attributi della relazione a cui viene applicato. Le tuple della relazione risultato vengono composte dalle tuple della relazione originale in modo che continuino ad essere un insieme in senso matematico.

TIME: viene applicato a due relazioni ed effettua il prodotto cartesiano delle tuple. Ogni tupla della prima relazione viene concatenata con ogni tupla della seconda.

JOIN: vengono concatenate le tuple di due relazioni in base al valore di un insieme dei loro attibuti.

UNION: applicando questo operatore a due relazioni compatibili, se ne ottiene una contenente le tuple di entrambe le relazioni. Due relazioni sono compatibili se hanno lo stesso numero di attributi e gli attributi corrispondenti nelle due relazioni hanno lo stesso dominio.

MINUS: applicato a due relazioni compatibili, ne restituisce una terza contenente le tuple che si trovano solo nella prima relazione.

INTERSECT: applicato a due relazioni compatibili, restituisce una relazione contenente le tuple che esistono in entrambe le relazioni.

DIVIDE: applicato a due relazioni che abbiano degli attributi comuni, ne restituisce una terza contenente tutte le tuple della prima relazione che possono essere fatte corrispondere a tutti i valori della seconda relazione.

Nelle seguenti tabelle, a titolo di esempio, sono raffigurati i risultati dell'applicazione di alcuni operatori relazionali alle relazioni Persone e Figli. Come nomi per le relazioni risultato si sono utilizzate le espressioni che le producono.

Persone

numero_persona	nome	cognome	data_nascita	sesso	stato_civile
2	Mario	Rossi	29/03/1965	M	Coniugato
1	Giuseppe	Russo	15/11/1972	M	Celibe
3	Alessandra	Mondella	13/06/1970	F	Nubile

Figli

numero_persona	nome_cognome	eta	sesso
2	Maria Rossi	3	F
2	Gianni Rossi	5	M

REStrICT (Persone)

sesso='M'

numero_persona	nome	cognome	data_nascita	sesso	stato_civile
2	Mario	Rossi	29/03/1965	M	Coniugato
1	Giuseppe	Russo	15/11/1972	M	Celibe

PROJECT sesso (Persone)

sesso
M
F

JOIN (Persone, Figli)

Persone(numero_persona)=Figli(numero_persona)

n.	nome	cognome	nascita	sesso	stato_civile	nome	eta'	sesso
2	Mario	Rossi	29/03/1965	M	Coniugato	Maria Rossi	3	F
2	Mario	Rossi	29/03/1965	M	Coniugato	Gianni Rossi	5	M

I database relazionali compiono tutte le operazioni sulle tabelle utilizzando l'algebra relazionale, anche se normalmente non permettono all'utente di utilizzarla. L'utente interagisce con il database attraverso un'interfaccia differente, il linguaggio SQL, un linguaggio dichiarativo che permette di descrivere insiemi di dati. Le istruzioni SQL vengono scomposte dal DBMS in una serie di operazioni relazionali.

3. Breve storia di SQL

La storia di SQL (che si pronuncia facendo lo spelling inglese delle lettere che lo compongono, e quindi "ess-chiu-el" e non "siquel" come si sente spesso) inizia nel 1974 con la definizione da parte di Donald Chamberlin e di altre persone che lavoravano presso i laboratori di ricerca dell'IBM di un linguaggio per la specificazione delle caratteristiche dei database che adottavano il modello relazionale. Questo linguaggio si chiamava SEQUEL (Structured English Query Language) e venne implementato in un prototipo chiamato SEQUEL-XRM fra il 1974 e il 1975. Le sperimentazioni con tale prototipo portarono fra il 1976 ed il 1977 ad una revisione del linguaggio (SEQUEL/2), che in seguito cambio' nome per motivi legali, diventando SQL. Il prototipo (System R) basato su questo linguaggio venne adottato ed utilizzato internamente da IBM e da alcuni sui clienti scelti. Grazie al successo di questo sistema, che non era ancora commercializzato, anche altre compagnie iniziarono a sviluppare i loro prodotti relazionali basati su SQL. A partire dal 1981 IBM comincio' a

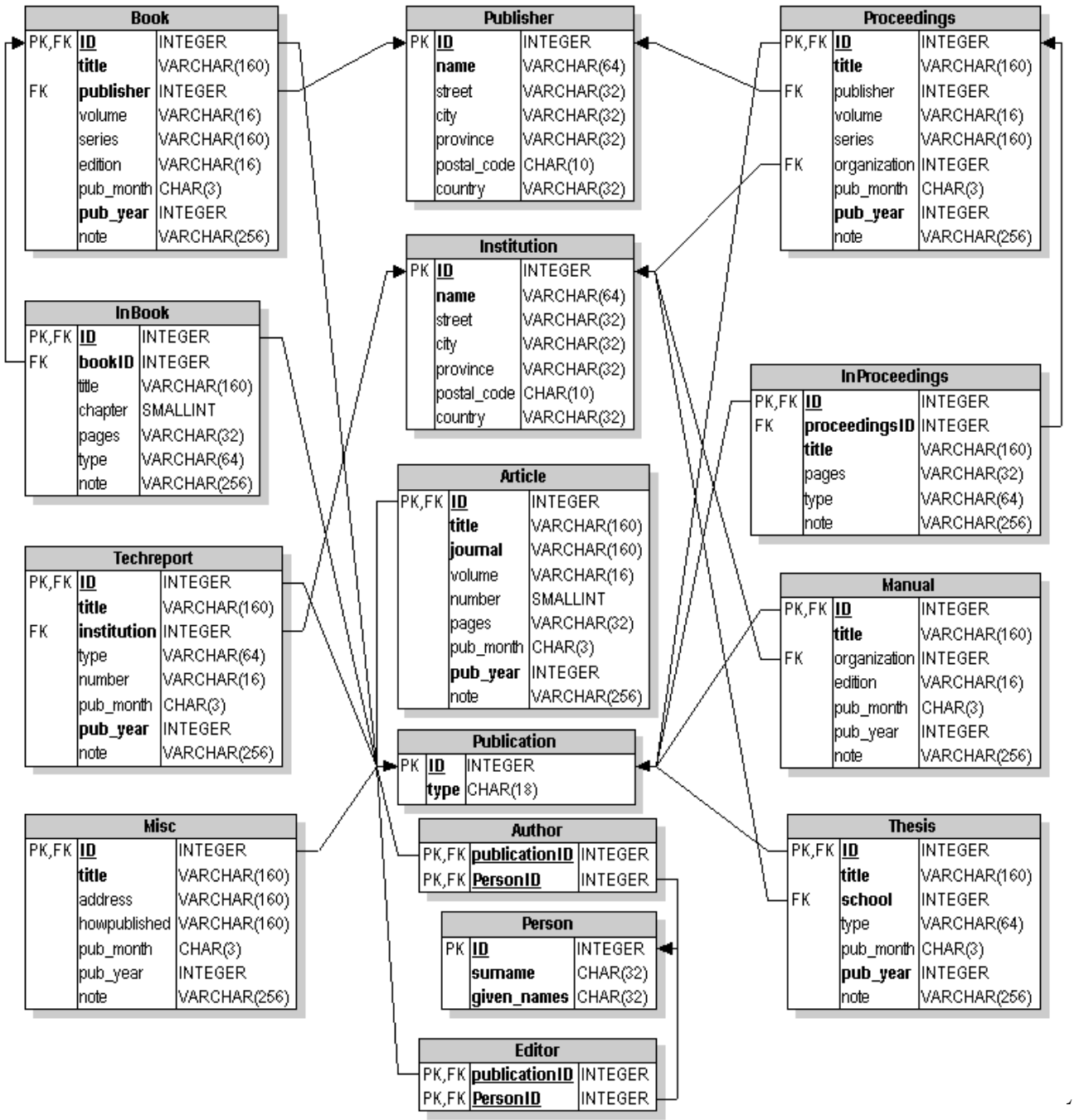
rilasciare i suoi prodotti relazionali e nel 1983 comincio' a vendere DB2. Nel corso degli anni ottanta numerose compagnie (ad esempio Oracle e Sybase, solo per citarne alcuni) commercializzarono prodotti basati su SQL, che divenne lo standard industriale di fatto per quanto riguarda i database relazionali.

Nel 1986 l'ANSI adotto' SQL (sostanzialmente adotto' il dialetto SQL di IBM) come standard per i linguaggi relazionali e nel 1987 esso divento' anche standard ISO. Questa versione dello standard va sotto il nome di SQL/86. Negli anni successivi esso ha subito varie revisioni che hanno portato prima alla versione SQL/89 e successivamente alla attuale SQL/92.
Il fatto di avere uno standard definito per un linguaggio per database relazionali, apre potenzialmente la strada alla intercomunicabilita' fra tutti i prodotti che si basano su di esso. Dal punto di vista pratico purtroppo le cose andarono differentemente. Infatti in generale ogni produttore adotta ed implementa nel proprio database solo il cuore del linguaggio SQL (il cosiddetto Entry level o al massimo l'Intermediate level), estendendolo in maniera proprietaria a seconda della propria visione del mondo dei database.

Attualmente e' in corso un processo di revisione del linguaggio da parte dei comitati ANSI e ISO, che dovrebbe portare alla definizione di cio' che al momento e' noto come SQL3. Le caratteristiche principali di questa nuova incarnazione di SQL dovrebbero essere la sua trasformazione in un linguaggio stand-alone (mentre ora viene usato come linguaggio ospitato in altri linguaggi) e l'introduzione di nuovi tipi di dato piu' complessi per permettere, ad esempio, il trattamento di dati multimediali.

4. Un database di esempio

Sara' ora presentata la struttura del database che verra' utilizzato per gli esempi delle successive lezioni. Non verranno descritte le fasi di analisi e i modelli concettuali e logici che sono stati necessari per giungere a tale struttura, dal momento che cio' andrebbe al di fuori degli scopi di questo corso. La struttura del database e' quella rappresentata nel diagramma relazionale di Figura 3. Ogni rettangolo rappresenta una relazione. Il nome della relazione e' contenuto nella sezione piu' scura nella parte alta del rettangolo. Il resto del rettangolo e' suddiviso in tre colonne, nelle quali sono definite le caratteristiche degli attributi che compongono la relazione. La colonna centrale contiene i nomi degli attributi, quella di destra il loro tipo (sono stati utilizzati i tipi dell'SQL/92) e quella di sinistra le loro proprieta'. Le proprieta' degli attributi sono indicate con le sigle "PK" e "FK", le quali significano rispettivamente che i corrispondenti attributi fanno parte della chiave primaria della relazione (Primary Key) o di una chiave esterna (Foreign Key). Le frecce congiungono appunto le chiavi esterne con le chiavi primarie a cui si riferiscono. I nomi degli attributi in neretto indicano che tali attributi non possono assumere il valore NULL, cioe' non possono essere indeterminati.



Lo scopo del database e' quello di contenere le informazioni bibliografiche di un insieme di pubblicazioni in modo da poterle consultare facilmente ed utilizzare per la costruzione di bibliografie anche estese. Esso e' stato modellato sulla falsa riga del sistema bibliografico del programma LaTeX, in modo da avere un ambiente ben consolidato a cui far riferimento e facilitare la realizzazione di programmi di conversione fra un sistema e l'altro.
Il significato delle relazioni che compongono il database e' il seguente:

Publication: Una generica pubblicazione. Normalmente questa relazione viene utilizzata solamente per assegnare un identificativo univoco a tutte le pubblicazioni presenti nel database, lasciando la specificazione delle altre caratteristiche in relazioni specifiche per ogni tipo di pubblicazione. Inoltre viene usata per implementare legami complessi fra le pubblicazioni e altre relazioni. Ad esempio quella fra una pubblicazione ed i suoi autori. Grazie alla struttura adottata si possono avere pubblicazioni scritte da piu' autori ed autori che scrivono piu' tipi di pubblicazioni.

Author: Rappresenta l'autore di una pubblicazione. La chiave primaria e' composta dall'identificativo della pubblicazione e da quello della persona; cio' garantisce l'unicita' dell'associazione fra le due entita'.

Editor: Rappresenta il curatore di una pubblicazione. La struttura e' identica a quella della tabella Author.

Person: Rappresenta una persona (ad esempio un autore) nel database. Attualmente le informazioni ritenute interessanti sono solo il cognome e il nome.

Publisher: La casa editrice di una pubblicazione.

Institution: L'istituzione (ad esempio un universita' o una software house) responsabile di una pubblicazione.

Book: Un libro con una precisa casa editrice.

InBook: Una parte di un libro. La parte puo' essere caratterizzata da un titolo, dal numero di capitolo o dal numero di pagina. Le informazioni riguardanti il libro ,e quindi comuni alle sue varie parti, vengono memorizzate nella relazione Book.

Proceedings: Gli atti di un congresso o di una conferenza.

InProceedings: Una parte degli atti di un congresso. Le informazioni riguardanti la pubblicazione che contiene la parte sono contenute nella relazione Proceedings.

Article: Un articolo pubblicato in un giornale o in una rivista.

Manual: Una pubblicazione di documentazione tecnica.

Techreport: Un rapporto tecnico pubblicato da una scuola o da un'altra istituzione.

Thesis: Una tesi di laurea o di dottorato.

Misc: Una pubblicazione che non rientra in nessuna delle precedenti categorie.

Non spieghero' il significato degli attributi che compongono le varie relazioni, dal momento che i loro nomi sono sufficientemente autoesplicativi. Un'unica annotazione sull'attributo "pub_month": esso e' stato definito di tipo CHAR(3), cioe' una stringa della lunghezza fissa di tre caratteri, e conterra' le abbreviazioni dei nomi dei mesi (le prime tre lettere dei nomi inglesi).

I legami fra le relazioni dovrebbero essere abbastanza semplici da capire. Come esempio per tutti verra' spiegata quello che collega la relazione Book con la relazione Publisher. Tale legame serve per descrivere la casa editrice di un libro. Nella relazione Book non sono presenti tutti i dati della casa editrice, ma solo un identificativo numerico per essa. Il numero sara' la chiave primaria della relazione Publisher e come tale permettera' di identificare una ben precisa casa editrice. Nella

relazione Book l'attributo publisher e' una chiave esterna verso la relazione Publisher.
Una situazione piu' complessa e' quella che coinvolge le relazioni Publication, Author e Person; infatti in Author sono presenti due chiavi esterne: una che identifica la pubblicazione a cui l'istanza di relazione si riferisce, e una che permette di risalire ai dati della persona che svolge il ruolo di autore. Ci si potrebbe chiedere quale sia l'utilita' della relazione Publication e perche' non si sia invece stabilito direttamente un legame fra la relazione Author e le relazioni che rappresentano i particolari tipi di pubblicazione. La risposta e' che il modello relazionale non permette di farlo. Infatti dal momento che un autore puo' scrivere piu' tipi di pubblicazioni l'attributo pubblicationID avrebbe dovuto essere una chiave esterna verso tutte le relazioni delle pubblicazioni, ma questo non e' permesso dal momento che contraddice la definizione stessa di chiave esterna.

5. Creare il database

Un database in un sistema relazionale e' composto da un'insieme di tabelle, che corrispondono alle relazioni del modello relazionale. Nella terminologia usata nell'SQL non si fa accenno alle relazioni, cosi' come non viene usato il termine attributo, ma viene usata la parola colonna, e non si parla di tupla, ma di riga. Nel seguito verranno usate indifferentemente le due terminologie, quindi tabella varra' per relazione, colonna per attributo, riga per tupla, e viceversa.
In pratica la creazione del database consiste nella creazione delle tabelle che lo compongono. In realta' prima di poter procedere alla creazione delle tabelle normalmente occorre creare in effetti il database, il che di solito significa definire uno sp azio dei nomi separato per ogni insieme di tabelle. In questo modo per un DBMS e' possibile gestire piu' database indipendenti contemporaneamente, senza che ci siano dei conflitti con i nomi che vengono utilizzati in ciascuno di essi. Il sistema previsto dallo standard per creare degli spazi dei nomi separati consiste nell'utilizzo dell'istruzione SQL "CREATE SCHEMA". Di solito tale sistema non viene utilizzato (o almeno non con gli scopi ed il significato previsti dallo standard), ma ogni DBMS prevede una procedura proprietaria per creare un database. Normalmente viene esteso il linguaggio SQL introducendo un'istruzione non prevista nello standard: "CREATE DATABASE".
La sintassi utilizzata da PostgreSQL, ma anche dai piu' diffusi DBMS, e' la seguente:

CREATE DATABASE nome_database

Con PostgreSQL e' anche disponibile un comando invocabile dalla shell Unix (o dalla shell del sistema utilizzato) che esegue la stessa operazione:

createdb nome_database

Per creare il nostro database bibliografico utilizzeremo quindi il comando:

createdb biblio

Una volta creato il database e' possibile creare le tabelle che lo compogono. L'istruzione SQL preposta a questo scopo e':

CREATE table nome_tabella (
nome_colonna tipo_colonna [clausola_default] [vincoli_di_colonna]
[, nome_colonna tipo_colonna [clausola_default] [vincoli_di_colonna] ...]
[, [vincolo_di tabella] ...])

nome_colonna: e' il nome della colonna che compone la tabella. Sarebbe meglio non esagerare con la lunghezza degli identificatori di colonna, dal momento che l'SQL Entry Level prevede nomi non piu' lunghi di 18 caratteri. Si consulti comunque la documentazione dello specifico database. I nomi devono iniziare con un carattere alfabetico.

tipo_colonna: e' l'indicazione del tipo di dato che la colonna potra' contenere. I principali tipi

previsti dallo standard SQL sono:

- CHARACTER(n)
  Una stringa a lunghezza fissa di esattamente n caratteri. CHARACTER puo' essere abbreviato con CHAR
- CHARACTER VARYING(n)
  Una stringa a lunghezza variabile di al massimo n caratteri. CHARACTER VARYING puo' essere abbreviato con VARCHAR o CHAR VARYING.
- INTEGER
  Un numero intero con segno. Puo' essere abbreviato con INT. La precisione, cioe' la grandezza del numero intero che puo' essere memorizzato in una colonna di questo tipo, dipende dall'implementazione del particolare DBMS.
- SMALLINT
  Un numero intero con segno con precisione non superiore a INTEGER.
- FLOAT(p)
  Un numero a virgola mobile, con precisione p. Il valore massimo di p dipende dall'implementazione del DBMS. E' possibile usare FLOAT senza indicazione della precisione, utilizzando quindi la precisione di default, anch'essa dipendente dall'implementazione. REAL e DOUBLE PRECISION sono dei sinonimi per un FLOAT con una particolare precisione. Anche in questo caso le precisioni dipendono dall'implementazione, con il vincolo che la precisione del primo non sia superiore a quella del secondo.
- DECIMAL(p,q)
  Un numero a virgola fissa di almeno p cifre e segno, con q cifre dopo la virgola. DEC e' un'abbreviazione per DECIMAL. DECIMAL(p) e' un'abbreviazione per DECIMAL(p,0). Il valore massimo di p dipende dall'implementazione.
- INTERVAL
  Un periodo di tempo (anni, mesi, giorni, ore, minuti, secondi e frazioni di secondo).
- DATE, TIME e TIMESTAMP
  Un preciso istante temporale. DATE permette di indicare l'anno, il mese e il giorno. Con TIME si possono specificare l'ora, i minuti e i secondi. TIMESTAMP e' la combinazione dei due precedenti. I secondi sono un numero con la virgola, permettendo cosi' di specificare anche frazioni di secondo.

clausola_default: indica il valore di default che assumera' la colonna se non gliene viene assegnato uno esplicitamente nel momento della creazione della riga. La sintassi da utilizzare e' la seguente:

DEFAULT { valore | NULL }

dove, valore e' un valore valido per il tipo con cui la colonna e' stata definita.

vincoli_di_colonna: sono vincoli di integrita' che vengono applicati al singolo attributo. Sono:

- NOT NULL, che indica che la colonna non puo' assumere il valore NULL.
- PRIMARY KEY, che indica che la colonna e' la chiave primaria della tabella.
- una definizione di riferimento, con cui si indica che la colonna e' una chiave esterna verso la tabella e i campi indicati nella definizione. La sintassi e' la seguente:

  REFERENCES nome_tabella [(colonna1 [, colonna2 ...])]
  [ON DELETE { CASCADE | SET DEFAULT | SET NULL }]
  [ON UPDATE { CASCADE | SET DEFAULT | SET NULL }]

  Le clausole ON DELETE e ON UPDATE indicano quale azione deve essere compiuta nel caso in cui una tupla nella tabella referenziata venga eliminata o aggiornata. Infatti in tali

casi nella colonna referenziante (che e' quella che si sta definendo) potrebbero esserci dei valori inconsistenti. Le azioni possono essere:

    - CASCADE: eliminare la tupla contenente la colonna referenziante (nel caso di ON DELETE) o aggiornare anche la colonna referenziante (nel caso di ON UPDATE).
    - SET DEFAULT: assegnare alla colonna referenziante il suo valore di default.
    - SET NULL: assegnare alla colonna referenziante il valore NULL.
- un controllo di valore, con il quale si permette o meno l'assegnazione di un valore alla colonna, in base al risultato di un espressione. La sintassi da usare e':

  CHECK (espressione_condizionale)

  dove espressione_condizionale e' un'espressione che restituisce vero o falso.
  Ad esempio, se stiamo definendo la colonna COLONNA1, definendo il seguente controllo:

  CHECK (COLONNA1 < 1000)

  in tale colonna potranno essere inseriti solo valori inferiori a 1000.

vincolo_di_tabella: sono vincoli di integrita' che possono riferirsi a piu' colonne della tabella. Sono:

- la definizione della chiave primaria:

  PRIMARY KEY (colonna1 [, colonna2 ...])

  Si noti che in questo caso, a differenza della definizione della chiave primaria come vincolo di colonna, essa puo' essere formata da piu' di un attributo.

- le definizioni delle chiavi esterne:

  FOREIGN KEY (colonna1 [, colonna2 ...]) definizione_di_riferimento

  La definizione_di_riferimento ha la stessa sintassi e significato di quella che puo' comparire come vincolo di colonna.

- un controllo di valore, con la stessa sintassi e significato di quello che puo' essere usato come vincolo di colonna.

Per chiarire meglio l'utilizzo dell'istruzione CREATE table, esaminiamo alcuni comandi che implementano il database bibliografico di esempio.

```
CREATE table Publication (
ID INTEGER PRIMARY KEY,
type CHAR(18) NOT NULL
);
```

La precedente istruzione crea la tabella Publication, formata dalle due colonne ID di tipo INTEGER, e type di tipo CHAR(18). ID e' la chiave primaria della relazione. Sull'attributo type e' posto un vincolo di non nullita'.

```
CREATE table Book (
ID INTEGER PRIMARY KEY REFERENCES Publication(ID),
title VARCHAR(160) NOT NULL,
publisher INTEGER NOT NULL REFERENCES Publisher(ID),
volume VARCHAR(16),
series VARCHAR(160),
edition VARCHAR(16),
pub_month CHAR(3),
```

pub_year INTEGER NOT NULL,
note VARCHAR(255)
);

Crea la relazione Book, formata da nove attributi. La chiave primaria e' l'attributo ID, che e' anche una chiave esterna verso la relazione Publication. Sugli attributi title, publisher e pub_year sono posti dei vincoli di non nullita'. Inoltre l'attributo publisher e' una chiave esterna verso la tabella Publisher.

CREATE table Author (
publicationID INTEGER REFERENCES Publication(ID),
personID INTEGER REFERENCES Person(ID),
PRIMARY KEY (publicationID, personID)
);

Crea la relazione Author, composta da due attributi: publicationID e personID. La chiave primaria in questo caso e' formata dalla combinazione dei due attributi, come indicato dal vincolo di tabella PRIMARY KEY. PublicationID e' una chiave esterna verso la relazione Publication, mentre personID lo e' verso la relazione Person.

6. INSERT, popolare il database

Col termine "popolazione del database" si intende l'attività di inserimento dei dati al suo interno. In un database relazionale ciò corrisponde alla creazione delle righe che compongono le tabelle che costituiscono il database. Normalmente la memorizzazione di una singola informazione corrisponde all'inserimento di una o più righe in una o più tabelle del database. Si prenda, ad esempio, la seguente informazione bibliografica:

`M. Agosti, L. Benfante, M. Melucci. OFAHIR: "On-the-Fly" Automatic Authoring of Hypertexts for Information Retrieval.In S. Spaccapietra, F. Maryansky (Eds), Searching for Semantics: Data Mining, Reverse Engineering. Proc. of the 7th IFIP 2.6 Working Conference on Database Semantics (DS-7), Leysin, Switzerland, October 1997, 129-154.`

Supponendo che nel database non sia gia' presente nessuna delle informazioni che la riguardano (come ad esempio qualcuno degli autori o gli atti del congresso a cui si riferisce), il suo inserimento nel nostro database d'esempio corrisponde all'inserimento delle seguenti righe:

- cinque righe nella tabella `Person` corrispondendi a ciascuno degli autori e dei curatori;
- una riga nella tabella `Institution`;
- due righe nella tabella `Publication`: una per gli atti del congresso e una per l'articolo contenuto in quegli atti;
- una riga nella tabella `Proceedings`;
- una riga nella tabella `InProceedings`;
- tre righe nella tabella `Author`, una per ciascun autore della pubblicazione;
- due righe nella tabella `Editor`, una per ciascun curatore della pubblicazione.

L'ordine delle precedenti operazioni non è puramente casuale, infatti l'inserimento delle righe deve essere fatto in modo da rispettare i vincoli imposti sulle tabelle. Ad esempio non potrà esistere una chiave esterna senza che sia stata prima inserita la riga a cui essa si riferisce, quindi prima di poter inserire una riga nella tabella `InProceedings` dovra' essere stata inserita la corrispondente riga nella tabella **Proceedings**.

Nel caso in cui un vincolo venga violato, il DBMS impedirà l'operazione di inserimento facendola fallire. Si veda la lezione precedente (Creare il database) per la descrizione dei vincoli che possono essere imposti su una tabella e sulle sue colonne.

L'istruzione SQL che effettua l'inserimento di una nuova riga in una tabella è **INSERT**. La sintassi con cui essa viene usata più comunemente è:

```
INSERT INTO nome_tabella [ ( elenco_campi ) ]
VALUES ( elenco_valori )
```

- *nome_tabella* è il nome della tabella in cui deve essere inserita la nuova riga.
- *elenco_campi* è l'elenco dei nomi dei campi a cui deve essere assegnato un valore, separati fra loro da una virgola. I campi non compresi nell'elenco assumeranno il loro valore di default o NULL se non hanno un valore di default.
  È un errore non inserire nell'elenco un campo che non abbia un valore di default e non possa assumere il valore NULL. Nel caso in cui l'elenco non venga specificato dovranno essere specificati i valori di tutti i campi della tabella.
- *elenco_valori* è l'elenco dei valori che verranno assegnati ai campi della tabella, nell'ordine e numero specificati dall'`elenco_campi` o in quello della definizione della tabella (se `elenco_campi` non viene specificato). I valori possono essere un'espressione scalare del tipo appropriato per il campo o le keyword **DEFAULT** o **NULL**, se il campo prevede un valore di default o ammette il valore NULL.

Il precedente esempio di inserimento viene quindi eseguito tramite le seguenti istruzioni SQL:

```
INSERT INTO Person VALUES ( 1, 'Agosti', 'Maristella' );
INSERT INTO Person VALUES ( 2, 'Benfante', 'Lucio' );
INSERT INTO Person VALUES ( 3, 'Melucci', 'Massimo' );
INSERT INTO Person VALUES ( 4, 'Spaccapietra', 'S.' );
INSERT INTO Person VALUES ( 5, 'Maryansky', 'F.' );

INSERT INTO Institution ( ID, name, city, country )
  VALUES ( 1, '7th IFIP 2.6 Working Conference on Database Semantics (DS-7)',
           'Leysin', 'Switzerland' );

INSERT INTO Publication VALUES ( 1, 'Proceedings' );
INSERT INTO Publication VALUES ( 2, 'InProceedings' );

INSERT INTO Proceedings ( ID, title, organization, pub_month, pub_year )
  VALUES ( 1, 'Searching for Semantics: Data Mining, Reverse Engineering',
           1, 'Oct', 1997 );

INSERT INTO InProceedings ( ID, proceedingsID, title, pages )
  VALUES ( 2, 1,
           'OFAHIR: "On-the-Fly" Automatic Authoring of Hypertexts for
Information Retrieval',
           '129-154' );

INSERT INTO Author VALUES ( 2, 1 );
INSERT INTO Author VALUES ( 2, 2 );
INSERT INTO Author VALUES ( 2, 3 );

INSERT INTO Editor VALUES ( 1, 4 );
INSERT INTO Editor VALUES ( 1, 5 );
```

Un'altra forma abbastanza utilizzata dell'istruzione **INSERT** segue la seguente sintassi:

```
INSERT INTO nome_tabella [ ( elenco_campi ) ]
```

```
istruzione_select
```

L'unica differenza con la precedente sintassi consiste nella sostituzione della clausola VALUES con un istruzione SELECT. Esamineremo questa istruzione nella prossima lezione, per il momento ci basti sapere che SELECT permette di estrarre dalle tabelle del database dei dati che vengono organizzati in una nuova relazione.

La precedente istruzione **INSERT** permette quindi di inserire nella tabella e nei campi specificati dati provenienti da altre tabelle. Ovviamente, affinché l'istruzione venga eseguita con successo, i dati prodotti dall'istruzione **SELECT** dovranno essere compatibili con i vincoli ed i domini dei campi della tabella in cui si sta effettuando l'inserimento.

7. SELECT, interrogare il database

Nella precedente lezione si sono esaminati i costrutti che il linguaggio SQL mette a disposizione per inserire i dati in un database relazionale. Vedremo ora invece le istruzioni che occorrono per estrarre da esso i dati che interessano. L'istruzione SQL preposta a tale scopo e' **SELECT**. Dal momento che l'interrogazione e' forse la funzionalita' piu' usata di un database, le opzioni che l'istruzione SELECT sono numerose e a volte piuttosto complicate. Per questo motivo esse verranno descritte in maniera semplificata, utilizzando invece degli esempi per la presentazione delle caratteristiche piu' complesse, in particolare quelle che riguardano la specificazione delle espressioni condizionali.
La sintassi con cui l'istruzione **SELECT** deve essere utilizzata e' la seguente:

SELECT [ALL | DISTINCT] lista_elementi_selezione
FROM lista_riferimenti_tabella
[WHERE espressione_condizionale]
[GROUP BY lista_colonne]
[HAVING espressione_condizionale]
[ORDER BY lista_colonne]

L'**istruzione SELECT** produce una tabella ottenuta applicando il seguente procedimento (per lo meno dal punto di vista logico, ogni DBMS ottimizza l'esecuzione delle interrogazioni secondo le proprie strategie):

- produce una tabella ottenuta come prodotto cartesiano delle tabelle specificate nella clausola FROM. Ogni elemento della lista_riferimenti_tabella segue la seguente sintassi:

  riferimento_tabella [[AS] alias_tabella]

  Il riferimento puo' essere il nome di una tabella o un' espressione (posta fra parentesi tonde) il cui risultato e' una tabella, quindi, ad esempio, anche un'altra SELECT. L'**alias** e' un nome che serve per indicare in breve un riferimento di tabella. Nel caso in cui il riferimento di tabella sia un espressione, e' obbligatorio specificare un alias.
- dalla tabella precedente elimina tutte le righe che non soddisfano l'espressione condizionale (cioe' le righe per cui l'espressione condizionale restituisce falso come risultato) della clausola **WHERE**.
- (se e' presente la clausola **GROUP BY**) le righe della tabella risultante dal passo 2 vengono raggruppate secondo i valori presenti nelle colonne specificate nella clausola GROUP BY. Righe con valori uguali vengono accorpate in un'unica riga. Le colonne non comprese nella clausola devono contenere delle espressioni con funzioni di aggregazione (come ad esempio AVG, che calcola la media) che quindi vengono calcolate producendo un singolo valore per ogni gruppo.
- (se e' presente la clausola HAVING) dal risultato del punto 3 vengono eliminate le righe che

non soddisfano l'espressione condizionale della clausola HAVING.

- Vengono calcolate le colonne presenti nella clausola SELECT (quelle nella lista_elementi_selezione). In particolare vengono calcolate le colonne con le funzioni di aggregazione derivanti dal raggruppamento avvenuto al punto 3. Ogni elemento della lista_elementi_selezione segue la seguente sintassi:

  espressione_scalare [[AS] alias_colonna]

  Un espressione scalare e' un'espressione che produce come risultato un valore scalare. I tipi di dato scalari del liguaggio SQL sono principalmente quelli descritti nella lezione 6 (Creare il database), esclusi INTERVAL, DATE, TIME e TIMESTAMP.
  Le espressioni scalari degli elementi della SELECT normalmente coinvolgono le colonne della tabella risultante dal punto 4. Nel caso in cui siano presenti delle ambiguita' a causa di colonne con nomi uguali in due o piu' tabelle comprese nella clausola FOR, esse possono essere risolte prefissando il nome o l'alias della colonna con il nome o l'alias della tabella, separati da un punto. Ad esempio, T.C indica la colonna C della tabella T. L'alias di colonna e' il nome che viene assegnato alla colonna.
  L'intero elenco delle colonne di una tabella puo' essere specificato utilizzando il carattere '*'.

- (se e' presente l'opzione DISTINCT) vengono eliminate le righe che risultano duplicate. Nel caso non sia presente ne' ALL ne' DISTINCT, viene assunto ALL.
- (se e' presente la clausola ORDER BY) le righe della tabella vengono ordinate secondo i valori presenti nelle colonne specificate nella clausola. La sintassi da utilizzare e' la seguente:

  ORDER BY nome_colonna [ASC | DESC] [, nome_colonna [ASC | DESC] ...]

L'ordinamento di default e' quello ascendente. Nel caso si desideri effettuare quello decrescente bisogna specificare l'opzione DESC. Se non viene specificata la clausola ORDER BY, la tabella e' da considerarsi senza un particolare ordinamento, infatti per la definizione di relazione del modello relazionale, le righe della tabella formano un insieme nel senso matematico e per gli elementi di un insieme non e' definita nessuna proprieta' di ordinamento. Nella pratica, invece, l'ordine che si ottiene non specificando la clausola di ordinamento è quasi sempre quello che rispecchia la loro memorizzazione fisica e quindi, di solito, quello secondo cui le righe sono state inserite nella tabella.

La sequenza di operazioni appena presentata deve essere considerata valida solo dal punto di vista concettuale. Infatti non e' detto che esse vengano eseguite esattamente in questo modo e in questo ordine, dal momento che ogni DBMS ottimizzerà le interrogazioni secondo le strategie piu' opportune.

Verranno ora esaminati alcuni esempi dell'istruzione **SELECT**.

ESEMPIO 1

```
SELECT surname FROM Person
ORDER BY surname
```

Estrae dalla tabella Person i cognomi e li ordina alfabeticamente. Nel nostro caso, il risultato e' il seguente:

```
surname
--------------------------------
Agosti
Batini
Bayer
```

Benfante
Carey
Cochowsky
DeWitt
Kim
Knuth
Lenzerini
Maryansky
McCreight
McGill
Melucci
Richardson
Salton
Santucci
Shekita
Spaccapietra

de Petra

Si noti l'errato ordinamento dell'ultima riga, dovuto al fatto che si e' usato il carattere ASCII minuscolo.
La precedente query restituirebbe righe duplicate nel caso in cui nella tabella fossero presenti persone con lo stesso cognome. Per evitare cio' occorre specificare l'opzione DISTINCT:

```
SELECT DISTINCT surname FROM Person
ORDER BY surname
```

ESEMPIO 2

```
SELECT * FROM Person
WHERE surname LIKE 'B%'
```

```
id|surname                         |given_names
--+--------------------------------+--------------------------------
 2|Benfante                        |Lucio
 6|Batini                          |Carlo
16|Bayer                           |R.
```

Produce una tabella avente tutte le colonne della tabella Person. Le righe vengono filtrate in modo che siano presenti solo quelle che hanno il cognome che inizia con il carattere 'B'. L'operatore LIKE permette un confronto fra stringhe di caratteri utilizzando dei pattern costruiti con i caratteri '%' e '_'. Il primo sostituisce un numero imprecisato di caratteri (anche 0), mentre il secondo ne sotituisce uno solo.

ESEMPIO 3

```
SELECT PUB.*, PER.surname AS S, PER.given_names
FROM Publication PUB, Author AUT, Person PER
WHERE PUB.ID = AUT.publicationID
AND AUT.personID = PER.ID
AND PUB.type = 'Book'
ORDER BY S
```

```
id|type              |s                                |given_names
--+------------------+---------------------------------+--------------
 3|Book              |Batini                           |Carlo
 7|Book              |Kim                              |Won
 8|Book              |Knuth                            |Donald E.
 9|Book              |Knuth                            |Donald E.
 3|Book              |Lenzerini                        |Maurizio
12|Book              |McGill                           |Michael J.
12|Book              |Salton                           |Gerard
 3|Book              |Santucci                         |Gaetano
 3|Book              |de Petra                         |Giulio
```

In questo caso la tabella risultante contiene tutte le colonne della tabella Publication (indicata con l'alias PUB definito nella clausola FROM) e le colonne surname e given_names della tabella Person. La clausola FROM genera il prodotto cartesiano delle tabelle Publication, Author e Person, di cui vengono selezionate solo le righe in cui l'identificativo della pubblicazione e quello dell'autore corrispondono. Inoltre ci si limita a considerare solo le pubblicazioni di tipo 'Book'. Per finire la tabella viene ordinata secondo il cognome dell'autore, indicato mediante l'alias S, definito nella clausola SELECT.

ESEMPIO 4

```
SELECT title, volume, pub_year
FROM Book
WHERE ID IN
( SELECT PUB.ID
FROM Publication PUB, Author AUT, Person PER
WHERE PUB.ID = AUT.publicationID
AND AUT.personID = PER.ID
AND PUB.type = 'Book'
AND PER.surname = 'Knuth' )
```

```
title                          |volume|pub_year
-------------------------------+------+--------
The Art of Computer Programming|     1|    1968
The Art of Computer Programming|     3|    1973
```

In questo esempio si vede l'utilizzo di un espressione condizionale che contiene l'operatore IN, il quale restitisce il valore vero se il valore dell'operando alla sua sinistra e' contenuto nella tabella risultato dell'espressione alla sua destra. La query fra parentesi produce una tabella di un'unica colonna contentente gli identificativi delle pubblicazioni di tipo 'Book' di cui Knuth e' authore. La query piu' esterna quindi estrae dalla tabella Book le informazioni dei libri con tali identificativi.

ESEMPIO 5

```
SELECT COUNT(*) FROM Publication count
-----
12
```

Conta il numero di righe presenti nella tabella Publication.

ESEMPIO 6

```
SELECT type, COUNT(ID) FROM Publication
GROUP BY type
```

type	count
Article	1
Book	6
InBook	1
InProceedings	1
Manual	1
Proceedings	1
Thesis	1

8. UPDATE, aggiornare il database

Normalmente le informazioni presenti in un database non sono statiche, ma evolvono nel tempo. C'e' quindi la necessita' non solo di aggiungere nuovi dati, ma di modificare quelli che sono gia' inseriti nelle tabelle del database. Le istruzioni SQL che vengono utilizzate a questo scopo sono UPDATE e DELETE. La prima modifica i valori presenti in una o piu' colonne di una o piu' righe di una tabella. La seconda elimina una o piu' righe di una tabella.
La sintassi di UPDATE e' la seguente:

UPDATE nome_tabella
SET elenco_assegnamenti
[WHERE espressione_condizionale]

Gli assegnamenti vengono specificati nella forma:

nome_colonna = espressione_scalare

L'istruzione UPDATE aggiorna le colonne della tabella che sono state specificate nella clausola SET, utilizzando i valori che vengono calcolati dalle corrispondenti espressioni scalari. Se viene espressa anche la clausola WHERE, vengono aggiornale solo le righe che soddisfano l'espressione condizionale. Si noti che l'espressione scalare utilizzata per aggiornare una colonna puo' anche essere il risultato di una query scalare, cioe' una query che restituisce una sola riga e una sola colonna.
Vediamo un esempio:

UPDATE Person
SET given_names = 'Stefano'
WHERE surname = 'Spaccapietra'

La precedente istruzione cambia il valore della colonna given_name della tabella Person nelle righe (nel nostro caso e' una sola) in cui la colonna surname ha valore 'Spaccapietra'.
La sintessi di DELETE e':

DELETE FROM nome_tabella
[WHERE espressione_condizionale]

L'istruzione delete elimina da una tabella tutte le righe che soddisfano l'espressione condizionale della clausola WHERE. Se WHERE non viene specificata, vengono cancellate tutte le righe della tabella.

Se nella definizione della tabella si sono specificate le clausole ON UPDATE o ON DELETE, nel momento in cui vengono eseguite queste operazioni viene eseguita anche l' azioni che era stata prevista sulle colonne referenziate (CASCADE, SET DEFAULT o SET NULL).

9. Modificare la struttura del database

Nel corso della precedente lezione si e' visto come modificare i dati gia' presenti nel database. A volte pero' non e' sufficiente modificare i dati, ma occorre aggiornare la struttura stessa del database per far si' che possano essere rappresentate nuove informazioni. Dal momento che la struttura del detabase e' data sostanzialmente dall'insieme delle tabelle che lo compongono, il suo aggiornamento corrisponde all'eliminazione di tabelle o alla modifica delle loro caratteristiche.
Per eliminare una tabella da un database il comando SQL da utilizzare e' DROP table:

DROP table nome_tabella { REStrICT | CASCADE }

nome_tabella e' il nome della tabella che deve essere eliminata.
Se si specifica la clausola CASCADE vengono automaticamente eliminati i vincoli di integrita' e le viste (view) in cui la tabella e' coinvolta. Viceversa, se si specifica la clausola REStrICT ed esistono dei vincoli di integrita' o delle viste che si riferiscono alla tabella, l'operazione fallisce.
Ad esempio, si siano definite le seguenti due tabelle:

CREATE table Prova1 (
Id INTEGER PRIMARY KEY,
Nome VARCHAR(50)) CREATE table Prova2 (
Id INTEGER PRIMARY KEY,
Nome VARCHAR(50),
toProva1 INTEGER REFERENCES Prova1(Id))

Volendo eliminare la tabella Prova1, l'istruzione:

DROP table Prova1 REStrICT

f

allira' dal momento che esiste un vincolo di integrita' che lega una chiave esterna della tabella Prova2 con la tabella Prova1.
Invece l'istruzione:

DROP table Prova1 CASCADE

verra' eseguita con successo e produrra' anche l'eliminazione del vincolo di integrita' referenziale presente sulla tabella Prova2.

Nel caso in cui si voglia modificare una tabella esistente nel database, l'istruzione da utilizzare e' ALTER table. Dal momento che la sintassi di questa istruzione e' piuttosto complessa, essa verra' spiegata scomponendola in base alle funzionalita' che si vogliono ottenere:

Aggiunta di una nuova colonna nella tabella

ALTER table nome_tabella ADD [COLUMN] definizione_colonna

nome_tabella e' il nome della tabella che si vuole modificare.
La definizione della colonna segue la stessa sintassi vista nella lezione "Creare il database" nella spiegazione dell'istruzione CREATE table. Dovranno quindi essere specificati il nome della colonna, il suo tipo ed eventualmente il suo valore di default e i vincoli imposti sulla colonna.
La parola chiave COLUMN puo' essere omessa (qui e in tutti i casi successivi).

Eliminazione di una colonna dalla tabella

ALTER table nome_tabella
DROP [COLUMN] nome_colonna { REStrICT | CASCADE }

nome_colonna e' il nome della colonna che si vuole eliminare. Le clausole RESStrIC e CASCADE

si comportano esattamente come nell'istruzione DROP table vista precedentemente.
L'istruzione fallira', oltre che nei casi gia' visti per REStrICT, se si tenta di eliminare una colonna che e' l'unica colonna di una tabella.

Modifica del valore di default di una colonna

ALTER table nome_tabella
ALTER [COLUMN] nome_colonna { SET clausola_default | DROP DEFAULT }

La sintassi ed il significato della clausola che definisce il nuovo valore di default sono identici a quelli della clausola_default che si usa nel comando CREATE table.

Eliminazione di un vincolo della tabella

ALTER table nome_tabella
DROP CONStrAINT nome_vincolo { REStrICT | CASCADE }

Elimina il vincolo identificato dal nome specificato. L'operazione fallisce se e' stata specificata la clausola REStrICT ed esistono altri vincoli che dipendono da quello che si intende eliminare.
Specificando la clausola CASCADE l'operazione verra' sempre completata con successo, cancellando inoltre i vincoli dipendenti da quello eliminato.
Spesso si vuole eliminare un vincolo a cui non e' stato assegnato un nome esplicitamente, facendo precedere la definizione del vincolo dalla clausola opzionale [CONStrAINT nome_vincolo]. In tal caso, dal momento che il DBMS avra' comunque assegnato un nome al vincolo per poterlo identificare, occorrera' interrogare le tabelle di sistema del DBMS ed ottenere da esse il suo nome. La particolare interrogazione richiesta dipende dallo specifico DBMS utilizzato.

Aggiunta di un vincolo alla tabella

ALTER table nome_colonna
ADD vincolo_di_tabella

La sintassi da utilizzare per la definizione del vincolo e' la stessa utilizzata nel comando CREATE table per i vincoli di tabella.

10. Utilizzo multiutente di un database

Fino ad ora abbiamo esaminato le caratteristiche del linguaggio SQL che riguardano la definizione e la manipolazione dei dati presenti in un database, senza preoccuparci del fatto che normalmente l'accesso a tali dati avviene in maniera concorrente da parte di piu' utenti contemporaneamente.
I meccanismi a sostegno di tale metodo di accesso riguardano principalmente la sicurezza dei dati, la gestione delle transazioni e la possibilita' di definire delle viste sulle tabelle del database.

1.1. Sicurezza

L'esecuzione di un operazione sui dati del database da parte di un utente e' subordinata al possesso da parte dell'utente dei necessari privilegi per la particolare operazione eseguita sullo specifico insieme di dati.
In generale i privilegi vengono attribuiti nella seguente maniera:

- Un utente che crea una tabella o qualunque altro oggetto del database ne e' il proprietario e gli vengono automaticamente garantiti tutti i privilegi applicabili a tale oggetto, con la possibilita' di impostare anche ad altri utenti tali privilegi (privilegio di concessione).
- Un utente che abbia un privilegio ed abbia in oltre su di esso il privilegio di concessione puo' assegnare tale privilegio ad un altro utente e passare ad esso anche il privilegio di

concessione.
- I privilegi sono concessi da chi ne abbia il permesso (cioe' dal proprietario dell'oggetto e da chi abbia il privilegio di concessione) mediante il comando GRANT e revocati mediante il comando REVOKE.

La sintassi del comando GRANT e' la seguente:

GRANT elenco_privilegi ON oggetto TO elenco_utenti [WITH GRANT OPTION]

Esso assegna all'utente i privilegi presenti nell'elenco_privilegi sull'oggetto specificato.
I privilegi assegnabili sono i seguenti (con le relative sintassi):

USAGE
Privilegio per usare uno specifico dominio o altro oggetto del database.

SELECT
Privilegio per accedere a tutte le colonne di una tabella o di una vista.

INSERT [(nome_colonna)]
Se viene specificata l'opzione nome_colonna, e' il privilegio per inserire valori nella colonna indicata di una tabella o di una vista. Senza il nome_colonna e' il privilegio per inserire valori in tutte le colonne, comprese quelle che saranno aggiunte in seguito.

UPDATE [(nome_colonna)]
Se viene specificata l'opzione nome_colonna, e' il privilegio per aggiornare il valore nella colonna indicata di una tabella o di una vista. In caso contrario permette di aggiornare il valore di tutte le colonne, comprese quelle che saranno aggiunte in seguito.

DELETE
Privilegio per eliminare righe da una tabella o da una vista.

REFERENCES [(nome_colonna)]
Se viene specificata l'opzione nome_colonna, e' il privilegio di riferirsi alla colonna indicata di una tabella o di una vista nella definizione di un vincolo di integrita'. Senza l'opzione concede tale privilegio per tutte le colonne, comprese quelle aggiunte in seguito.

L'oggetto a cui il privilegio si riferisce e' generalmente una tabella o una vista. La sintassi per la sua specificazione e' in tal caso:

[table] nome_tabella
Nel caso di altri oggetti segue la sintassi:
tipo_oggetto nome_oggetto

dove tipo_oggetto puo' essere DOMAIN, CHARACTER SET, COLLATION o trANSLATION (si veda C.J. Date - "A Guide to The SQL Standard" per una spiegazione di tali oggetti).
Nel caso di oggetti diversi da tabelle o viste, l'unico privilegio applicabile e' quello di USAGE.

L'elenco_utenti e' un elenco di identificativi di utenti o gruppi di utenti. Puo' anche essere utilizzata la parola chiave PUBLIC, che indica tutti gli utenti e i gruppi conosciuti nel sistema.

Se e' presente l'opzione [WITH GRANT OPTION], viene assegnato inoltre il privilegio di concessione, che permette agli utenti di trasferire ulteriormente i privilegi loro assegnati.

Ad esempio:

GRANT SELECT, INSERT, UPDATE(nome) ON persona TO benfante WITH GRANT OPTION

assegna all'utente benfante i privilegi di SELECT e INSERT su tutte le colonne della tabella persona e quello di UPDATE sulla sola colonna nome di tale tabella. Gli viene inoltre garantito il privilegio di assegnare tali permessi ad altri utenti.

Per togliere i privilegi agli utenti si usa invece REVOKE:

REVOKE [GRANT OPTION FOR] elenco_privilegi ON oggetto FROM elenco_utenti { REStrIC | CASCADE }

elenco_privilegi, oggetto ed elenco_utenti hanno le stesso significato delle corrispondenti opzioni di GRANT. L'opzione GRANT OPTION FOR revoca il privilegio di concessione. Se viene specificata la clausola REStrICT, il comando REVOKE puo' fallire nel caso in cui l'utente a cui vengono revocati i privilegi li abbia ulteriormente concessi ad altri utenti. Se e' presente invece la clausola CASCADE, l'istruzione verra' sempre completata con successo e verranno revocati i privilegi anche di quegli utenti e di tutti gli utenti a cui essi li hanno concessi (...e cosi' via, finche' non ci saranno piu' permessi "abbandonati", cioe' concessi senza che chi li ha concessi ne sia ancora in possesso). Verranno inoltre distrutti gli oggetti del database costruiti grazie a tali permessi.

2.2. Gestione delle transazioni

Le transazioni SQL sono insiemi di istruzioni che devono essere trattati come delle unita' atomiche, cioe' non scomponibili nelle singole istruzioni da cui sono formate. Grazie a tale atomicita' le transazioni permettono di eseguire operazioni complesse sul database mantenendone l'integrita'. Infatti una transazione viene eseguita con successo se e solo se tutte le operazioni che la compongono terminano con successo. In caso contrario, cioe' se una delle operazioni fallisce o se la transazione viene esplicitamente annullata, tutte le operazioni precedenti vengono annullate anch'esse. Le operazioni di una transazione non hanno alcun effetto sul database fino a quando la transazione non viene completata con successo.

Dal momento che ad un database possono accedere piu' utenti contemporanamente, in ogni istante potremmo avere piu' transazioni che manipolano il database in maniera concorrente. Lo standard SQL prevede che normalmente le transazioni debbano essere eseguite nel "livello di isolamento serializzabile" (isolation level SERIALIZABLE), cioe' in una modalita' di esecuzione che garantisca la "serializzabilita'" delle transazioni. Il fatto che le transazioni siano serializzabili significa che il loro effetto complessivo sul database e' quello che si otterrebbe se esse venissero eseguite in maniera non concorrente l'una di seguito all'altra.
Nel linguaggio SQL standard non esiste un'istruzione che faccia iniziare esplicitamente una transazione. Le istruzioni vengono divise in due classi: quelle che possono iniziare una transazione e quelle che non la fanno iniziare. Nel momento in cui si cerca di eseguire un'istruzione della prima classe, se non e' gia' in corso una transazione, ne viene cominciata una. La transazione continua fino a quando una delle istruzioni fallisce, causando l'annullamento dell'intera transazione, o se vengono eseguite le istruzioni COMMIT WORK o ROLLBACK WORK.

L'istruzione COMMIT WORK termina la transazione confermandola, rendendo quindi definitivi gli effetti delle sue istruzioni sul database. L'istruzione ROLLBACK WORK invece la termina annullandola.
Spesso i DBMS che si trovano in commercio implementano la gestione delle transazioni in maniera differente da quanto previsto dallo standard (almeno nelle loro impostazioni di default). In tal caso, di solito e' previsto un comando che inizia esplicitamente una transazione (BEGIN trANSACTION, START WORK, o altro). Se una transazione non e' stata iniziata esplicitamente, le singole istruzioni ne compongono una ciascuna.
Per capire meglio quali potrebbero essere le conseguenze della manipolazione concorrente dei dati di un database senza l'utilizzo delle transazioni, vediamone un'esempio. Supponiamo di avere un database con il quale gestiamo gli ordini dei prodotti che vendiamo. In particolare, quando un cliente ci sottopone una richiesta per un prodotto, ne verifichiamo la disponibilita' e nel caso in cui possiamo soddisfare l'ordine, sottraiamo alla quantita' in giacenza la quantita' che ci e' stata

richiesta. traducendo tutto cio' in SQL, otteniamo la quantita' in giacenza con l'istruzione (istruzione A):

```
SELECT giacenza FROM prodotti
WHERE prodottoID=1453
```

L'aggiornamento della giacenza, una volta verificata la disponibilita', e' ottenuta dalla seguente istruzione (istruzione B):

```
UPDATE prodotti
SET giacenza=giacenza-1
WHERE prodottoID=1453
```

Se due utenti cercano di eseguire questa operazione, senza che le due istruzioni che la compongono siano state raggruppate in una transazione, potrebbe accadere che le istruzioni vengano eseguite nell'ordine e con i risultati seguenti :

- Istruzione A eseguita dall'utente 1: viene restituita una giacenza del prodotto pari a 1, quindi l'ordine verra' approvato.
- Istruzione A eseguita dall'utente 2: come prima la giacenza e' 1 e anche in questo caso l'ordine verra' approvato.
- Istruzione B eseguita dall'utente 1: a questo punto nel database la giacenza per il prodotto vale 0.
- Istruzione B eseguita dall'utente 2: ora la giacenza vale -1, che e' ovviamente un valore errato.

Come si vede il risultato finale e' che uno dei due clienti non potra' ricevere (almeno non subito) la merce, dato che non ne avevamo in giacenza una quantita' sufficiente per entrambi i clienti. Se le due istruzioni fossero state inserite in una transazione, il problema non sarebbe sorto, dato che la transazione del secondo utente non avrebbe potuto leggere il valore della giacenza fino a quando non fosse stata completata la transazione del primo utente. A quel punto, la giacenza avrebbe avuto valore 0 e l'ordine non sarebbe stato erratamente approvato.

3.3. Viste

Fino ad ora le uniche tabelle con cui abbiamo avuto a che fare sono state quelle definite con il comando CREATE table. Il linguaggio SQL mette anche a disposizione la possibilita' di definire delle tabelle "virtuali", le viste, calcolate a partire da altre tabelle. Esse sono virtuali nel senso che non occupano spazio su disco, ma sono il risultato di interrogazioni su altre tabelle e quindi sempre allineate con i valori contenuti in tali tabelle.
L'istruzione SQL per definire una vista e' la seguente:

```
CREATE VIEW nome_vista [ ( elenco_nomi_colonne ) ]
AS espressione_tabella
```

Essa crea una vista chiamata nome_vista definita dall'espressione_tabella. Tipicamente espressione_tabella e' un'instruzione select che produrra' la tabella che interessa.
l'elenco_nomi_colonne puo' essere usata per assegnare dei nomi alle colonne della vista. Cio' e' utile nel caso in cui le colonne derivanti dall'espressione tabella siano il risultato di un calcolo (ad esempio COUNT(nome_colonna)) e non abbiano quindi un nome esplicito. Una volta creata, una vista puo' essere utilizzata come una normale tabella. Le uniche limitazioni riguardano le operazioni che modificano i dati in essa contenuti. Infatti, non tutte le viste sono aggiornabili. Le regole che discriminano fra una vista aggiornabile e una non aggiornabile sono piuttosto complesse e non e' questa la sede per descriverle (si vedano i libri in bibliografia, in particolare quello di C.J. Date).

Qui ci limiteremo a cercare di capire, mediante un esempio, perche' questo accade.
Proviamo ad utilizzare la seguente vista sul nostro database bibliografico:

```
CREATE VIEW book_publisher89
AS SELECT B.title, P.name
FROM Book B, Publisher P
WHERE B.publisher = P.ID
AND B.pub_year=1989
```

Essa ci permette di eseguire la query che la definisce semplicemente utilizzando l'istruzione:

```
SELECT * FROM book_publisher89
```

Possiamo anche impostare ulteriori condizioni (o fare in modo che il risultato sia ordinato secondo una particolare colonna della vista, ecc...):

```
SELECT title FROM book_publisher89
WHERE name = "ACM Press"
```

Quest'ultima interrogazione ci fornisce l'elenco dei titoli dei libri pubblicati dall'ACM Press nel 1989.

Come si vede per quanto riguarda operazioni di interrogazione una vista si comprta come una normale tabella. Le differenze sorgono quando si cerca di applicare ad una vista delle operazioni di aggiornamento. Ad esempio, se cerchiamo di eseguire la seguente istruzione:

```
INSERT INTO book_publisher89
VALUES( "Nuovo libro", "publisher")
```

Il DBMS non riuscira' ad eseguirla, restituendo un errore tipo "No INSERT permission". Il motivo e' che non e' in grado di creare le righe corrispondendi al nostro nuovo record nelle due tabelle "reali" da cui la vista e' originata (i problemi sono vari: deve creare solo una righa nella tabella Book e collegarla ad una particolare riga nella tabella Publisher, o creare una riga in entrambe le tabelle; come decidere quali valori assegnare alle chiavi primarie degli eventuali nuovi record; quali valori assegnare agli altri campi delle due tabelle; ecc...)

Grazie alle viste (e all'assegnazione oculata dei permessi agli utenti) e' possibile fare in modo che utenti diversi abbiano una percezione della struttura del database anche molto diversa da quella che ha realmente e impedire che particolari categorie di utenti possano accedere ad informazioni che non competono loro.
Ad esempio, supponiamo di avere una tabella in cui sono memorizzati i dati anagrafici dei dipendenti di una ditta e l'ammontare del loro stipendio mensile. Ovviamente si vorrebbe impedire la consultazione dei dati relativi agli stipendi a tutti gli utenti, eccetto a quelli che si devono occupare dell'erogazione/amministrazione degli stessi. Un sistema per fare cio' e' quello di definire una vista contenente solo le colonne dei dati anagrafici. In questo modo tutti gli utenti autorizzati ad accedere ai dati anagrafici, ma non a quelli degli stipendi, lo potranno fare solo attraverso tale vista. Ulteriori partizionamenti potrebbero essere fatti in senso orizzontale, creando ad esempio un vista che contiene solo le informazioni sui dirigenti ed una che contiene i dati degli altri dipendenti. Inoltre, le viste spesso contribuiscono a facilitare quella indipendenza fra applicazioni e struttura dei dati, che rende i database degli strumenti tanto utili. Infatti se ad un certo punto fosse necessario cambiare la struttura del database (scomponendo, ad esempio, una tabella in due tabelle per motivi di efficienza), non sarebbe necessario modificare tutte le applicazioni adattandole alla nuova struttura, ma sarebbe sufficiente creare delle opportune view, in modo che dal punto di vista delle applicazioni niente sia stato cambiato.